

## Crea la estructura correcta del proyecto y se instalan correctamente las dependencias con Composer.

Para instalar y crear correctamente el proyecto, creo una carpeta en htdocs llamada tema5. Luego accedo a ella por línea de comandos y tecleo composer init:

```
C:\xampp\htdocs>cd tema5  
C:\xampp\htdocs\tema5>composer init
```

Después he ido contestando a las preguntas que se me iban planteando:

```
This command will guide you through creating your composer.json config.  
Package name (<vendor>/<name>) [denitsa/tema5]: usuario/usuario  
Description []: ejercicio tema 5  
Author [, n to skip]: usuario <usuario@example.com>  
Minimum Stability []:  
Package Type (e.g. library, project, metapackage, composer-plugin) []: project  
License []: GPL  
  
Define your dependencies.  
  
Would you like to define your dependencies (require) interactively [yes]?
```

Hasta que llegó la parte de instalar las dependencias necesarias. Aquí tuve bastantes problemas, ya que al instalar x versión, me daba conflictos ya sea con fazinotto o con milon barcode. Después de buscar por internet y ver bastantes tutoriales, encontré que la mejor manera es la siguiente:

Me salto la parte de añadir las dependencias desde aquí y genero el fichero .json:

```

Loading composer repositories with package information
Updating dependencies
Lock file operations: 11 installs, 0 updates, 0 removals
- Locking doctrine/inflector (1.4.3)
- Locking illuminate/contracts (v5.8.36)
- Locking illuminate/support (v5.8.36)
- Locking milon/barcode (5.1.0)
- Locking nesbot/carbon (2.43.0)
- Locking psr/container (1.0.0)
- Locking psr/simple-cache (1.0.1)
- Locking symfony/polyfill-mbstring (v1.22.0)
- Locking symfony/polyfill-php80 (v1.22.0)
- Locking symfony/translation (v5.2.1)
- Locking symfony/translation-contracts (v2.3.0)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 11 installs, 0 updates, 0 removals
- Downloading milon/barcode (5.1.0)
- Installing doctrine/inflector (1.4.3): Extracting archive
- Installing psr/simple-cache (1.0.1): Extracting archive
- Installing psr/container (1.0.0): Extracting archive
- Installing illuminate/contracts (v5.8.36): Extracting archive
- Installing symfony/translation-contracts (v2.3.0): Extracting archive
- Installing symfony/polyfill-php80 (v1.22.0): Extracting archive
- Installing symfony/polyfill-mbstring (v1.22.0): Extracting archive
- Installing symfony/translation (v5.2.1): Extracting archive
- Installing nesbot/carbon (2.43.0): Extracting archive
- Installing illuminate/support (v5.8.36): Extracting archive
- Installing milon/barcode (5.1.0): Extracting archive
0 package suggestions were added by new dependencies, use `composer suggest` to see details.

```

Una vez hecho esto, dentro de la carpeta tema5, creo manualmente las carpetas cache, public, src y views donde cada una contendrá cache, los ficheros php, las clases y las vistas respectivamente.

Después, abro el proyecto desde visual studio y abro el fichero .json, añadiendo las dependencias sin indicar ninguna versión en concreto, dejando un asterisco para evitar conflictos entre versiones. Añado así el autoloader y le indico qué clases usar (las de la carpeta src:)

**\*\***La única versión que he dejado indicada es la de philo blade, es la manera que encontré para que funciona sin errores

```

{} composer.json > ...
1  {}
2      "name": "usuario/usuario",
3      "description": "p",
4      "type": "project",
5      "require": {
6          "philo/laravel-blade": "3.1",
7          "milon/barcode": "*",
8          "fzaninotto/faker" : "*"
9      },
10
11     "config": {
12         "optimize-autoloader": true
13     },
14     "autoload": {
15         "psr-4": {
16             "Classes\\": "src"
17         }
18     },
19     "license": "GPL",
20     "authors": [
21         {
22             "name": "usuario",
23             "email": "usuario@ejemplo.com"
24         }
25     ]

```

Después de esto, accedo a la consola y dentro de tema5 ejecuto composer update para reflejar los cambios hechos:

```

C:\xampp\htdocs\tema5>composer update
Loading composer repositories with package information
Updating dependencies
Lock file operations: 1 install, 0 updates, 0 removals
- Locking milon/barcode (6.0.4)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
- Downloading milon/barcode (6.0.4)
- Installing milon/barcode (6.0.4): Extracting archive
Package fzaninotto/faker is abandoned, you should avoid using it. No replacement was suggested.
Generating optimized autoload files
9 packages you are using are looking for funding.
Use the `composer fund` command to find out more!

C:\xampp\htdocs\tema5>

```

Después accedo a la carpeta vendor>composer y abro el fichero autoload\_psr4.php:

Me dirijo al final del fichero:

```
'Faker\\' => array($vendorDir . '/fzaninotto/faker/src/Faker'),  
'Doctrine\\Inflector\\' => array($vendorDir . '/doctrine/inflector/lib/Doctrine/Inflector'),  
'Doctrine\\Common\\Inflector\\' => array($vendorDir . '/doctrine/inflector/lib/Doctrine/Common/Inflector'),  
'Classes\\' => array($baseDir . '/src'),  
'Carbon\\' => array($vendorDir . '/nesbot/carbon/src/Carbon'),  
'providers' => [Milon\Barcode\BarcodeServiceProvider::class,],  
'aliases' => [  
    'DNS1D' => Milon\Barcode\Facades\DNS1DFacade::class,  
    'DNS2D' => Milon\Barcode\Facades\DNS2DFacade::class,  
]
```

Me aseguro que la línea Clases\\ => array(\$baseDir . '/src'), esté añadida, porque de no ser así no nos funcionaría autoload correctamente.

Para terminar de configurar las librerías añado las líneas que marco en naranja y vuelvo a ejecutar composer update dentro de la terminal. Sé que no es exactamente la manera que viene indicada en los apuntes para configurarlo, pero es la única que me funciona sin errores

Se crean correctamente, con los métodos necesarios, las clases "Conexion" y "Jugador" y se hace uso de "namespaces" y herencia.

Dentro de la carpeta src he metido las clases Conexión y Jugador, las cuales contienen los siguientes métodos: (ambas pertenecen al namespace Clases)

Clase conexión:

```
<?php
//Creo una clase conexion que extiende de PDO
namespace Clases;
use PDO;
use PDOException;
class Conexion extends PDO {
    //Indico los datos para hacer la conexión
    private $tipo_de_base = 'mysql';
    private $host = '127.0.0.1';
    private $nombre_de_base = 'practicaunidad5';
    private $usuario = 'gestor';
    private $contrasena = 'secreto';

    //Sobreescribo el método constructor de PDO
    public function __construct() {
        try{
            parent::__construct("{${this->tipo_de_base}:dbname={${this->nombre_de_base}};host={${this->host}};charset=utf8",
            ${this->usuario}, ${this->contrasena});
        }catch(PDOException $e){
            //Manejo de errores
            echo 'Ha surgido un error y no se puede conectar a la base de datos. Detalle: ' . $e->getMessage();
            exit;
        }
    }
}
```

## Clase Jugador:

Para esta he creado un constructor con parámetros, getters, setters, un toString(), un método para insertar jugadores , un método que comprueba si existe un jugador, un método que comprueba si existe un barcode, un método para listar jugadores y un método para pintar un código de barras en formato html pasándole la numeración. (Estos dos últimos no los termino usando)

```
<?php
namespace Clases;
use Clases\Conexion;
use PDO;
use PDOException;
use \Milon\Barcode\DNS1D;

class Jugador{
    private $nombre;
    private $apellidos;
    private $posicion;
    private $dorsal;
    private $codigoBarras;

    //Constructor
    public function __construct($n, $a, $p,$d,$codBarras){
        $this->nombre=$n;
        $this ->apellidos=$a;
        $this->posicion=$p;
        $this ->dorsal=$d;
        $this ->codigoBarras=$codBarras;
    }
}
```

```
public function getNombre(){  
    return $this->nombre;  
}  
  
public function getApellidos(){  
    return $this->apellidos;  
}  
  
public function getPosicion(){  
    return $this->posicion;  
}  
  
public function getDorsal(){  
    return $this->dorsal;  
}  
  
public function getCodigoBarras(){  
    return $this->codigoBarras;  
}  
  
public function setNombre($n){  
    $this->nombre = $n;  
}  
  
public function setApellidos($a){  
    $this->apellidos = $a;  
}  
  
public function setPosicion($p){  
    $this->posicion = $p;  
}
```

```
public function setDorsal($d){  
    $this->dorsal = $d;  
}  
  
public function setCodigoBarras($c){  
    $this->codigoBarras = $c;  
}
```

```

//Comprueba si existe un jugador, lo utilizo sobretodo antes de insertar para evitar duplicados.
//Le paso como parámetro el dorsal, que es único por cada jugador y no se repite
public static function existeJugador($dorsal){
    $conexion = new Conexion();
    $existe = false;
    $resultado = $conexion->query("SELECT * FROM jugadores where dorsal = $dorsal");
    //Si el jugador existe se mete al while y pongo el booleano a true
    while ($registro = $resultado->fetch()) {
        $existe = true;
    }

    return $existe;
}

```

```

/*
Compruebo si existe el barcode que le paso en la base de datos. Lo utilizo antes de insertar
porque no pueden existir dos jugadores con el mismo barcode
*/
public static function existeBarcode($barcode){
    $conexion = new Conexion();
    $existe = false;
    $resultado = $conexion->query("SELECT * FROM jugadores where barcode = $barcode");
    //Si el jugador existe se mete al while y pongo el booleano a true
    while ($registro = $resultado->fetch()) {
        $existe = true;
    }

    return $existe;
}

```

Para el método insertar retorno una variable llamada resultado. Primero compruebo si existe el jugador, si ya existe en la base de datos, resultado = 2, después compruebo si existe el barcode, entonces resultado = 1. Si no se da ninguna de estas condiciones inserto el jugador y pongo resultado = 0, al final de la función lo retorno.



```

//Para insertar el jugador
public function insertarJugador(Jugador $jug)
{
    $resultado = -1; //Variable que retorno para saber si se ha insertado o se ha producido error
    //Si no existe el jugador ni el barcode, lo inserto
    if (!Jugador::existeJugador($jug->getDorsal())) {
        if (!Jugador::existeBarcode(($jug->getCodigoBarras()))) {
            $conexion = new Conexion();
            //Configuro el atributo para que me muestre los errores sql
            $conexion->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
            try {
                //Abro la transacción y preparo la consulta
                $conexion->beginTransaction();
                $stmt = $conexion->prepare("INSERT INTO jugadores (nombre, apellidos, dorsal, posicion, barcode
VALUES (:nombre, :apellidos, :dorsal, :posicion, :barcode )");
                // Bind
                $stmt->execute([
                    'nombre' => $jug->getNombre(),
                    'apellidos' => $jug->getApellidos(),
                    'dorsal' => $jug->getDorsal(),
                    'posicion' => $jug->getPosicion(),
                    'barcode' => $jug->getCodigoBarras()
                ]);
                $resultado = 0; //Resultado = 0 es que la inserción se ha hecho bien
                $conexion->commit();
            } catch (PDOException $e) {
                //Manejo de errores
                echo 'Ha surgido un error y no se puede realizar la inserción. Detalle: ' . $e->getMessage();
            }
        } else {
            $resultado = 1; //Devuelve 1 si ya existe el codigo de barras

```

```

        }
    } else {
        $resultado = 1; //Devuelve 1 si ya existe el codigo de barras
        //echo "Ya existe el codigo de barras";
    }
} else {
    $resultado = 2; //Devuelve 2 si el jugador ya existe en la bd
    //echo "El jugador ya existe";
}

return $resultado;
}

```

```

//Para pintar el codigo de barras del usuario con el nº de dorsal que le pasamos
public static function pintaCodigobarras($dorsal){
    $conexion = new Conexion();
    $cod= '';
    $resultado = $conexion->query("SELECT barcode FROM jugadores where dorsal = $dorsal");
    //Si el producto existe se mete al while y pongo el booleano a true
    while ($registro = $resultado->fetch()) {
        $cod = $registro[0];
    }

    //Objeto DNS1D, le paso el barcode obtenido de la base de datos y lo imprimo con echo
    $d = new DNS1D();
    $d->setStorPath(__DIR__.'/cache/');
    echo $d->getBarcodeHTML($cod, 'EAN13');
}

//Devuelve un listado con todos los jugadores que existan
public function listadoJugadores(){
    $conexion = new Conexion();
    $existe = false;
    $resultado = $conexion->query("SELECT * FROM jugadores");
    while ($registro = $resultado->fetch()) {
        echo $registro[0], $registro[1], $registro[2],$registro[3],$registro[4], Jugador::pintaCodigobarras($registro[3]);
    }
}

```

```

//Devuelve en formato String los atributos del objeto
public function __toString(){
    echo $this->nombre ;
    echo "<br>";
    echo $this->apellidos ;
    echo "<br>";
    echo $this->posicion;
    echo "<br>";
    echo $this->dorsal;
    echo "<br>";
    echo $this ->codigoBarras;
}

```

Respecto a la herencia, la he utilizado para las vistas. Tengo una plantilla general o principal, la cual van heredando todas las vistas que voy creando:

Plantilla principal:

```
> plantillas > plantilla1.blade.php > html > head > meta
<!--Plantilla base que voy a usar en todas las vistas -->
<!doctype html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
        content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <!-- css para usar Bootstrap -->
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" integrity="s
    <!-- Fontawesome CDN -->
    <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.3.1/css/all.css"
        integrity="sha384-mzrmE5qonljUremFsqc01SB46JvROS7bZs3IO2EmfFsd15uHvIt+Y8vEf7N7fWAU" crossorigin="anonymous">
<title>@yield('titulo')</title>
</head>
<body style="background:#FFA07A">
<div class="container mt-3">
    <h3 class="text-center mt-3 mb-3">@yield('encabezado')</h3>
    @yield('contenido')
</div>
</body>
</html>
```

Como esto es algo común a todas las vistas, lo dejo en esta plantilla principal para no repetir código. Posteriormente, en las vistas hago esto:

```
<!--Vista de crear jugadores. Extiende de la plantilla principal, plantilla1-->
@extends('plantillas.plantilla1')
@section('titulo')
    {{$titulo}}
@endsection
@section('encabezado')
    {{$encabezado}}
@endsection
@section('contenido')
```

**Crea correctamente los datos de Ejemplo con "fzaninotto/faker".**

Tengo la vista vinstalacion:

```
<!--Vista muy sencilla, contiene el botón de instalar que lleva a crearDatos.php. Ahí genero
datos aleatorios que luego inserto en la base de datos -->
@extends('plantillas.plantilla1')
@section('titulo')
{{$titulo}}
@endsection
@section('encabezado')
{{$encabezado}}
@endsection
@section('contenido')

<form name="formulario1" action="crearDatos.php" method="POST">
    <div class="container">
        <div class="col-xs-3 text-center">
            <form action="metodos.php" method="POST">
                <input type="submit" value="Instalar datos de ejemplo" name="instalar" class="btn btn-success">
            </form>
        </div>
    </div>
</form>
@endsection
```

La cargo desde el archivo instalación.php:

```
<?php
session_start();
require '../vendor/autoload.php';

use Philo\Blade\Blade;

$views = '../views';
$cache = '../cache';
$blade = new Blade($views,$cache);

    $titulo = 'Instalación';
    $encabezado = 'Instalación';
    echo $blade
        ->view()
        ->make('vinstalacion', compact('titulo', 'encabezado'))
        ->render();
?>
```

Pero lo importante es el action del formulario, podemos ver que es crearDatos.php. Desde este fichero es donde genero datos aleatorios:

Declaro lo necesario para la clase y cargo las librerías que voy a usar:

```
<?php
require '../vendor/autoload.php';

use Clases\Jugador;
use Philo\Blade\Blade;
use \Milon\Barcode\DNS1D;
$views = '../views';
$cache = '../cache';
$faker = Faker\Factory::create(); //Instancio objeto de Fzaninotto faker
$valores = array(); //Creo array para guardar los nº aleatorios para dorsal que genero. Evita duplicados
$x = 0; //Variable de control para el método que genera nºs aleatorios para dorsales
```

Creo los siguientes métodos que voy a utilizar para crear 10 jugadores con datos aleatorios:

```
//Devuelve un nº aleatorio entre 1 y 6 equivalente al enum de la base de datos para la posicion
function posicionAleatoria(){
    $d=rand(1,6);
    return $d ;
}

//Genera un nº de dorsal aleatorio del 1 al 10 evitando que se repitan
function dorsalAleatorio(){
    global $valores;
    global $x;
    while ($x<10) { //Mientras x sea menor que 10 (el nº de jugadores)
        $num_aleatorio = rand(1,10); //Creo nº aleatorio
        if (!in_array($num_aleatorio,$valores)) { //Compruebo si está en el array, si no lo está lo añado
            array_push($valores,$num_aleatorio);
            $x++; //Incremento x para la siguiente vuelta
            return $num_aleatorio; //Retorno un nº aleatorio del 1 al 10 sin repetir
        }
    }
}
```

```
//Genera un código de barras para cada jugador
function codigoBarras(){
    //Con este bucle for genero un nº aleatorio de 13 cifras
    $digitos = '';
    for($i = 0; $i < 12; $i++){
        $digitos .= mt_rand(0,9);
    }

    //CÓDIGO QUE HE USADO PARA PROBAR A IMPRIMIRLO, NO LO UTILIZO AQUÍ ES UNA SIMPLE PRUEBA
    /*Creo el objeto DNS1D, usado por Milon Barcode, le añado el path de la caché
    $d = new DNS1D();
    $d->setStorPath(__DIR__.'/cache/');
    echo $d->getBarcodeHTML($digitos, 'EAN13'); Para imprimir el código de barras*/
    return $digitos; //Retorno la numeración
}
```

Finalmente con un bucle for los voy creando, haciendo uso de fzaninotto faker , milon barcode y los métodos que acabo de describir:

```
//Para crear 10 jugadores con datos aleatorios
for ($i = 0; $i <= 9; $i++) {
    //Creo un objeto del tipo Jugador y le inserto atributos aleatorios
    $jugadores=(new Jugador($faker->firstname,$faker->lastname, posicionAleatoria(),dorsalAleatorio(), codigoBarras()));
    $jugadores->insertarJugador($jugadores); //Inserto el jugador en la base de datos
}
```

Posteriormente redirijo al usuario a la página jugadores.php la cual le mostrará los jugadores que acaba de crear en forma de listado:

```
echo "<script>alert('Se han instalado los datos de prueba. Redirigiendo a jugadores...')
window.location.href=\"jugadores.php\";
</script>";
```

Las páginas de la carpeta **"public"** cargan las vistas apropiadamente y les pasan los parámetros necesarios.

Dada la vista vcrear con todos sus elementos html, la cargo desde fcrear.php:

```
<?php
require '../vendor/autoload.php';
use \Milon\Barcode\DNS1D;
use Philo\Blade\Blade;

$view = 'vcrear';
$cache = '../cache';
$blade = new Blade($view, $cache);

$titulo = 'Crear jugador';
$encabezado = 'Crear jugador';
echo $blade
    ->view()
    ->make($view, compact('titulo', 'encabezado'))
    ->render();
?>
```

Resultado:

### Crear jugador

Nombre

Apellidos

Dorsal

Posición

Portero

Código de barras

Crear

Limpiar

Volver

Generar código

Aquí debo de añadir que intenté colocar los 3 inputs de debajo bien alineados pero después de intentar con distintos elementos de bootstrap, no conseguí dar con ello. Pero tuve en cuenta los requisitos, como que el código de barras sea ReadOnly así como controlar la longitud de dorsal para evitar que salten errores de Mysql al insertar.

Para la vista vinstalacion:

La cargo desde instalación.php:

```
<?php
session_start();
require '../vendor/autoload.php';

use Philo\Blade\Blade;

$views = '../views';
$cache = '../cache';
$blade = new Blade($views,$cache);

    $titulo = 'Instalación';
    $encabezado ='Instalación';
    echo $blade
        ->view()
        ->make('vinstalacion', compact('titulo', 'encabezado'))
        ->render();
?>
```

Resultado:

# Instalación

Instalar datos de ejemplo

Para la vista vjugadores la cargo desde jugadores.php:

```
?php
require '../vendor/autoload.php';

use Clases\Jugador;
use Clases\Conexion;
use \Milon\Barcode\DNS1D;
use Philo\Blade\Blade;

$view = '../views';
$cache = '../cache';
$blade = new Blade($view, $cache);
$pinta = (new pintaDatosJugadores())->listadoJugadores();
$titulo = 'Listado de jugadores';
$encabezado = 'Listado de jugadores';





echo $blade
    ->view()
    ->make('vjugadores', compact('titulo', 'encabezado', 'pinta'))
    ->render();

/*
```

Resultado:

## Listado de jugadores

Nuevo jugador

Nombre completo	Posición	Dorsal	Código de barras
Gerson Kunze	Central	1	
Frankie Ziemann	Portero	2	
Wilburn Abshire	Delantero	3	
Jennie Davis	Portero	4	



## El formulario para crear un jugador funciona correctamente, controlando posibles errores.

Para controlar errores, lo primero que hice fue que todos los campos del form sean required. Después, en el campo de dorsal establecí la siguiente propiedad:

```
</div>  
<input type="number" min="1" max="999999999"  
</div>
```

Porque si se introduce un número más grande que ese, salta un error de Mysql porque el dato es de tipo INT en la base de datos, y solo puede almacenar 4 bytes.

Para manejar los datos, he usado las variables de sesión, ya que al generar un código (y ser este un enlace) se pierden. En el fichero crearJugador es donde establezco todo lo necesario para que el formulario funcione:

```
//RECOJO TODOS LOS DATOS DEL FORMULARIO EN VARIABLES DE SESIÓN PARA CONSERVAR SUS DATOS AUNQUE  
//SE PULSE SOBRE EL ENLACE QUE GENERA UN CÓDIGO  
if (isset($_POST['nombre'])) {  
    $_SESSION['sesNombre'] = $_POST['nombre'];  
}  
  
if (isset($_POST['apellidos'])) {  
    $_SESSION['sesApellidos'] = $_POST['apellidos'];  
}  
  
if (isset($_POST['dorsal'])) {  
    $_SESSION['sesDorsal'] = $_POST['dorsal'];  
}  
  
if (isset($_POST['posicion'])) {  
    $_SESSION['sesPosicion'] = $_POST['posicion'];  
}  
  
if(isset($_POST['volverDeCrear'])){  
    session_unset();  
    header('Location: jugadores.php');  
}
```

```

//Si recibo la variable crear
if (isset($_POST['crear'])) {
    //Compruebo si hay algún barcode en la variable de sesión, de no ser así aviso al usuario para que genere uno
    if(isset($_SESSION['codigoGenerado'])){
        //Creo un objeto del tipo jugador con los datos recogidos de las variables de sesión
        $jugadores = (new Jugador(
            $_SESSION['sesNombre'],
            $_SESSION['sesApellidos'],
            $_SESSION['sesPosicion'],
            $_SESSION['sesDorsal'],
            $_SESSION['codigoGenerado']
        ));

        //En resultado guardo lo que me devuelve el método insertarJugador
        $resultado = $jugadores->insertarJugador($jugadores);
    }
}

```

Recordemos que el método insertarJugador devuelve un entero que nos indica si hay algún error o de lo contrario se ha insertado bien:

```

//Si me devuelve 0, significa que lo ha insertado bien. Muestro alert al usuario y borro
//los datos de las variables de sesión. Vuelvo a fcrear.php para ver el formulario
if($resultado == 0){
    echo "<script> alert('Jugador insertado');
    window.location.href=\"fcrear.php\";
    </script>";
    session_unset();
}

/*
Si resultado = 1 significa que el barcode generado ya existe en la base de datos,
le muestro alert al usuario para que genere otro y lo redirijo al formulario tambien
*/
if($resultado == 1){
    echo "<script> alert('El barcode generado ya existe, por favor genere otro');
    window.location.href=\"fcrear.php\";
    </script>";
}

/*
Si resultado = 2 significa que el jugador que intenta insertar ya existe en la base de datos.
Muestro alert al usuario indicándolo y vuelvo a redirigirlo a la página del formulario
*/
if($resultado == 2){
    echo "<script> alert('El jugador que intenta insertar ya existe.');
    window.location.href=\"fcrear.php\";
    </script>";
}

```

Si se ha insertado todo bien, reseteo los datos guardados en las variables de sesión para la siguiente inserción.

Esto lo utilizo en caso de que se haya pulsado crear con todos los datos rellenos pero no se haya generado un barcode:

```
}else{
    echo "<script> alert('Genere un código para el jugador');
    window.location.href=\"fcrear.php\";
    </script>";
}
```

### Se muestran y generan correctamente los códigos de barra.

En la clase generarCode.php tengo el siguiente método que comprueba si el que se ha generado ya existe en la base de datos:

```
//Comprueba si el barcode que le pasamos ya existe en la base de datos
function existeCodBarras($barcode){
    global $conexion;
    $existe = false;
    $resultado = $conexion->query("SELECT barcode FROM jugadores where barcode = $barcode");
    while ($registro = $resultado->fetch()) {
        $existe = true;
    }
}
```

Cada vez que el usuario pulsa sobre generar código desde el formulario de crear, se ejecuta lo siguiente:

```
/*
    Genero un bucle infinito. Esto lo hago para que siga generando códigos mientras exista en la base de datos. Es decir, si encuentra uno que ya existe, genera otro. Si el nuevo se vuelve a encontrar en la base de datos, se vuelve a generar. En caso de que genere uno que no exista, se guarda en la variable de sesión y se redirige a fcrear.php de nuevo.
*/
while(true){
    //Genera un código de barras para cada jugador
    $digitos = '';
    for ($i = 0; $i < 12; $i++) {
        $digitos .= mt_rand(0, 9);
    }
    //Si no existe el codigo generado en la base de datos, lo meto en una variable de sesión para
    //poder operar con el en otras páginas y redirijo al usuario a la página que carga la vista de
    //Crear un usuario nuevo. Hago break y salgo del bucle
    if (!existeCodBarras($digitos)) {
        $_SESSION['codigoGenerado'] = $digitos;
        echo $digitos;
        header('Location: fcrear.php');
        break;
    }else{
        echo "El código de barras ya existe";
        continue;
    }
}
```

El barcode generado lo almaceno en una variable de sesión para poder operar con él en otras páginas.

En crearJugador.php tengo el siguiente método:

```
//Pinta el código de barras en el formulario de crear jugador
function pintaCodBarras($cod)
{
    if(isset($_SESSION['codigoGenerado']))
    {
        $d = new DNS1D();
        $d->setStorPath(__DIR__ . '/cache/');
        echo $d->getBarcodeHTML($cod, 'EAN13');
    }else{
        echo "";
    }
}
```

Y dentro del input del formulario llamo a dicho método:

```
<!--En este campo pinto el código que he generado. Dicho código lo almaceno en una variable de sesión llamada codigoGenerado, de manera
que puedo operar con él a través de las distintas páginas. En este input, compruebo si dicha variable de sesión contiene datos, y si los
tiene, llamo al método pintaCodBarras creado en crearJugador.php, el cual se encarga de pintarlo en formato HTML. -->
<input type="text" readonly class="form-control" id="codBarras" name="codBarras" required value=<?php
    if (isset($_SESSION['codigoGenerado'])) {
        pintaCodBarras($_SESSION['codigoGenerado']);
    }
?>>
```

## Las vistas hacen uso de una plantilla común.

```
plantilla1.blade.php X
views > plantillas > plantilla1.blade.php > html > head > meta
1 <!--Plantilla base que voy a usar en todas las vistas -->
2 <!doctype html>
3 <html lang="es">
4 <head>
5     <meta charset="UTF-8">
6     <meta name="viewport"
7         content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
8     <meta http-equiv="X-UA-Compatible" content="ie=edge">
9     <!-- css para usar Bootstrap -->
10    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" integrity="s
11    <!--Fontawesome CDN-->
12    <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.3.1/css/all.css"
13        integrity="sha384-mzrmE5qonljUremFsqc01SB46JvROS7bZs3IO2EmfFsd15uHvIt+Y8vEf7N7fWAU" crossorigin="anonymous">
14 <title>@yield('titulo')</title>
15 </head>
16 <body style="background:#FFA07A">
17 <div class="container mt-3">
18     <h3 class="text-center mt-3 mb-3">@yield('encabezado')</h3>
19     @yield('contenido')
20 </div>
21 </body>
22 </html>
```

Ejemplo de plantilla que la hereda y la usa:

```
vcrear.blade.php X
views > vcrear.blade.php > ...
1 <!--Vista de crear jugadores. Extiende de la plantilla principal, plantilla1-->
2 @extends('plantillas.plantilla1')
3 @section('titulo')
4     {{$titulo}}
5 @endsection
6 @section('encabezado')
7     {{$encabezado}}
8 @endsection
9 @section('contenido')
10
11 <?php
12 require_once('crearJugador.php');
13 ?>
14 <!--Manejo los datos en un formulario el cuál manda la info por POST. Lo mando fcrear.php donde proceso
15 los datos-->
16 <br>
17 <br>
18 <br>
19 <form name="formulario1" action="crearJugador.php" method="POST">
20 <div class="container">
```

**La página de inicio comprueba si existen datos o no en la tabla "jugadores" y hace las redirecciones adecuadas.**

En index.php ejecuto lo siguiente:

```
?php
require '../vendor/autoload.php';
use Clases\Jugador;
use Clases\Conexion;

$conexion = new Conexion();

//Si existe registros devuelve true, llevo al usuario a la tabla donde se muestran
if(existeRegistros()){
    header('Location:jugadores.php');
}else{
    header('Location:instalacion.php');
    //De lo contrario lo llevo a la pagina instalacion para crear datos de prueba
}

//Compruebo si la tabla tiene datos. Si me devuelve 0 es que no tiene registros
function existeRegistros(){
    global $conexion;
    $existe = false;
    $resultado = $conexion->query("SELECT COUNT(*) FROM jugadores");
    //Si el producto existe se mete al while y pongo el booleano a true
    while ($registro = $resultado->fetch()) {
        if($registro[0] != 0) {
            $existe = true;
        }
    }
    return $existe;
}
```

Tengo un método que comprueba si existen registros en la tabla jugadores, devuelve true o false en función de lo que encuentre. Si devuelve true, redirige al usuario a jugadores.php donde verá el listado. De lo contrario lo lleva a instalación.php donde se crearán 10 jugadores aleatorios.

### **Se hace uso correcto de las sesiones para mostrar mensajes.**

En mi caso, más que utilizarla para mostrar mensajes (me gusta más mostrar un alert al usuario), lo he visto como una herramienta útil para operar con los datos a través de las distintas páginas. Un ejemplo de ello es como guardo el barcode generado en generarCode.php y después lo utilizo en la vista así como en crearJugador.php. Por otra parte como he comentado, guardo los datos del formulario en variables de sesión también, evitando que se pierdan.

### **El diseño es adecuado y el código está comentado.**

Se han incluido comentarios lo más extendidos y redactados posible para explicar bien lo realizado.

\*\*Cargué la librería para Font awesome icons pero por alguna razón no se mostraban dentro de los botones, aún así he intentado cuidar el diseño lo máximo posible